API Usability

Steven Clarke
Usability Engineer

Visual Studio Usability Group

What Is a Usable API?

- Absolute minimum is one that users can be successful with
- Better is one that users can be successful with and *can understand*
- Even better is one that users can be successful with, can understand, and can *predict*

Why Bother About Usability?

- Allows developers to focus on their application
- Reduces errors that developers might make
- Increases the "learnability" of your framework
- Helps you to make better decisions about design
- Helps you to design the right thing, and design the thing right

How to Design Usable APIs

- Design for usability from the start
  - Know what to pay attention to
  - Use the Cognitive Dimensions
- Understand your users
  - Use the Cognitive Dimensions
- Understand their scenarios
  - Scenario-based design
- Get feedback early and often
  - Usability studies

Cognitive Dimensions (CDs)

- A set of 12 dimensions that measure
  - Usability aspects of an API
  - Developer requirements
- Provide a means to talk about an API from an end-user perspective
- Simple and effective framework
- http://weblogs.asp.net/stevencl/

1. Abstraction Level
2. Learning Style
3. Working Framework
4. Work-Step Unit
5. Progressive Evaluation
6. Premature Commitment
7. Penetrability
8. API Elaboration
9. API Viscosity
10. Consistency
11. Role Expressiveness
12. Domain Correspondence

Using CDs to Describe an API

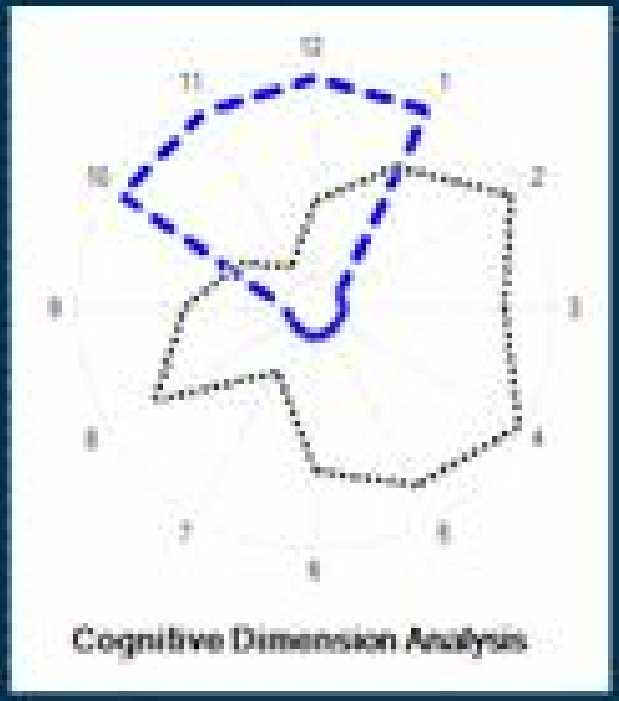
- Abstraction Level
  - What types of abstraction does the API expose?
- Learning Style
  - How can the API be learned?
- Working Framework
  - What is the mental "working set" imposed on developers by the API?
- Work-Step Unit
  - How much work does the developer need to do to accomplish a task?
- Premature Commitment
  - To what extent does the developer need to make decisions about how to use the API?
- Progressive Evaluation
  - To what extent can progress towards completion of a task be evaluated?

Using CDs to Describe Users

- Abstraction Level
  - Aggregate, factored, and primitive components
- Learning Style
  - Learn by doing, learn before doing
- Working Framework
  - Local scope, global scope
- Work-Step Unit
  - Incremental, parallel
- Premature Commitment
  - Minimal, extensive
- Progressive Evaluation
  - Incremental, parallel

Using CDs to Evaluate an API

- Gather usability data
- Describe data with CDs and compare to what is known about your users



Gathering User Feedback

- Early and often
- API review
  - Review code samples and API headers
- Experience review
  - Application-building and CD framework to review
- Usability study
  - Don't have to wait until API is implemented

API Review

- Review code samples
  - Review from perspective of writing and reading code
  - Is it likely your users will write this code?
  - Is it likely that they will understand it?
  - Consider each aspect of the framework
- Review headers
  - Review from perspective of browsing API
  - Look for role-expressiveness issues
  - Is it clear what the role of each component is?
- Can be performed before a line of code has been implemented
- Will capture surface and obvious issues
  - Subtle usability issues are more difficult to capture
  - User model discrepancies unlikely to surface

Experience Review

- Group of developers use API to build some application
- Perform structured interview with team
  - For each
- Works best when developers have same characteristics as users
- Good way of gathering long-term data
- Less control over the areas of the API that you gather feedback on
- Could be expensive

Usability Study

- Determine the feedback you need to gather
- Recruit a representative group of users
- Design tasks for each user to attempt
  - Focused tasks
  - End-to-end tasks
- Cheaper than experience review
- Can be performed before the API is completely implemented

What Is Involved

- Be prepared!
- Work with your usability engineer
- Provide UE with set of scenarios to study
- UE prepares study and task list, and recruits participants
- UE runs study
  - Team attends as many sessions as possible
- UE analyzes data and presents results
- Address issues, repeat process

Common Usability Problems

- Lack of expressive names
  - Properties, parameters, methods, classes
- Lack of expressive types
- Implementation-centric APIs vs. task-centric
  - `GetNavigator` instead of `RunQuery`
- Poor domain correspondence
- Factory/builder design pattern
- Lack of consistency
- Unclear dependencies between components
- Over-reliance on documentation